esp-idf-matter と Rust でサクッと Matter デバイスを作ってみる

Matter のプロトコル・スタックの実装としては、Matter の規格を管理している **CSA (Connectivity Standards Alliance)** によるリファレンス実装 **connectedhomeip** があります。このほかにも TypeScript による実装として **matter.js** [*1] Rust による実装として **rs-matter** [*2] があります。

今回は rs-matter を用いて、簡単な Matter デバイスを作成してみます。

対象のデバイス AtomS3 Lite

Matter デバイスのハードウェアとしては、M5Stack AtomS3 Lite [*3] および ENV.IV Sensor Unit [*4] を用います。AtomS3 Lite は、M5Stack が製造・販売する無線通信機能付きのプロトタイプ用モジュールです。Espressif の無線通信機能付き MCU **ESP32-S3** を搭載しています。また GROVE 互換ポートにより各種センサなどを追加できます。

今回は温湿度および気圧計センサ・ユニットである ENV.IV Sensor Unit を追加し、温湿度を計測できるようにします。

ESP32-S3 向け Rust 開発環境の構築

ESP32-S3 向けのソフトウェアを Rust で開発するには、公式の Rust ツールチェインに加えて、Espressif の fork 版の Rust ツールチェインをインストールします。ESP32-S3 に搭載されている CPU である Xtensa LX7 に対応した Rust ツールチェインは現在のところ公式で用意されておらず、Espressif の fork 版が必要なためです。

まずは公式のツールチェインを **rustup (https://rustup.rs/)** を用いてインストールしておきます。セットアップ手順は rustup の公式手順を参照してください。

次に、Espressif の fork 版 Rust ツールチェインをインストールします。まずは Espressif の Rust ツールチェインを管理する **espup (https://github.com/esp-rs/espup)** コマンドを `cargo install` でインストールします。

```
cargo install espup
```

インストールが完了したら、Espressif のツールチェインの動作に必要な環境変数の定義が `~/export-esp.sh` に生成されるので読みこんでおきます。(Windows は不要)

```
source ~/export-esp.sh
```

これで ESP32-S3 向けの Rust 開発環境が用意できました。

Matter テストツールのセットアップ

Matter デバイスのコミッショニングを試すために、まずは connectedhomeip をビルドします。connectedhomeip のリポジトリにあるドキュメント[*5]の手順にしたがってビルドします。

```
sudo apt-get install git gcc g++ pkg-config libssl-dev libdbus-1-dev \
  libglib2.0-dev libavahi-client-dev ninja-build python3-venv python3-dev \
  python3-pip unzip libgirepository1.0-dev libcairo2-dev libreadline-dev
git clone https://github.com/project-chip/connectedhomeip -b v1.3.0.0 --depth 1 --recurse-submodules connectedhomeip_v1.3.0.0
cd connectedhomeip_v1.3.0.0
./scripts/examples/gn_build_example.sh examples/chip-tool $PWD/out/linux
```

`connectedhomeip_v1.3.0.0/out/linux/chip-tool` に Matter のテストツールである `chip-tool` のバイナリが生成されます。

esp-idf-matter のサンプルを試す

esp-idf-matter の Matter デバイスサンプルをビルドします。執筆時点の HEAD は `7dabdba8d6f308b582cb2e4532634cbebc4c1c7f` です。もしうまく行かない場合はこのリビジョンを試してみてください。

```
git clone https://github.com/ivmarkov/esp-idf-matter
# うまく行かない場合はリビジョンを指定してチェックアウト
# git checkout 7dabdba8d6f308b582cb2e4532634cbebc4c1c7f
```

以下のコマンドでサンプルをビルドします。

```
MCU=esp32s3 cargo build --target xtensa-esp32s3-espidf --example light --features examples
```

ビルドが成功したら、Atom S3 を PC に接続してビルドしたファームウェアを書き込みます。

```
MCU=esp32s3 cargo run --target xtensa-esp32s3-espidf --example light --features examples
```

ファームウェアを書き込んだら AtomS3 のログが表示された状態になるので、別のターミナル等で先ほどビルドした chip-tool を使って AtomS3 を Matter デバイスとしてコミッショニングします。

```
cd out/linux
./chip-tool pairing ble-wifi 1 (WiFiアクセスポイントのSSID) (WiFiアクセスポイントのパスワード) 20840 20202021
```

成功すれば

```
CHIP:CTL: Successfully finished commissioning step 'Cleanup'
CHIP:TOO: Device commissioning completed with success
```

のように `Device commissioning completed with success` というメッセージが出力されます。執筆時点のサンプルはタスクのメモリ割り当てサイズが足りないようで不安定です。成功しない場合は一旦無視して次に進みます。

esp-idf-matter のサンプルの改造

esp-idf-matter の `light` サンプルは On Off Light Device Type の Matter デバイスの実装です。LED やボタンと言った外付けのハードウェアは使用しておらず、5 秒ごとに勝手にライトをトグルする動作を繰り返します。これだけだと Matter デバイスとしては面白くないので、AtomS3Lite 上の RGB LED を実際の On Off Light Device Type のオン・オフ状態に対応するようにしま

[*1] https://github.com/project-chip/matter.js
[*2] https://github.com/project-chip/rs-matter
[*3] https://www.switch-science.com/products/8670?srsltid=AfmBOooqXzwKnRlvVfqXchNUW5Dd1MFLF4FCLqpkAioo5kmrtaju-JAKET
[*4] https://docs.m5stack.com/ja/unit/ENV%E2%85%A3%20Unit
[*5] https://github.com/project-chip/connectedhomeip/blob/v1.3.0.0/docs/guides/BUILDING.md

す。また、プッシュ・ボタンを押すと現在の点灯・消灯状態をトグルするようにします。

さらに、外付けの ENV.IV Sensor Unit に搭載されている SHT40 から温度・湿度を取得して Matter の Temperature Sensor Device Type および Humidity Sensor Device Type として公開します。現時点ではこれらの実装に必要な Temperature Measurement Cluster および Relative Humidity Measurement Cluster は実装されていないため、実装する必要があります。

Temperature Measurement Cluster/Relative Humidity Measurement Cluster は Attribute を 3 つ持つだけの比較的単純なクラスターですので実装してしまいます。Temperature Measurement Cluster の実装の一部を示します。

```
pub const ID: u32 = 0x0402; // Temperature Measurement ClusterのID (Application Cluster Specより)
#[derive(FromRepr, EnumDiscriminants)]
#[repr(u16)]
pub enum Attributes {
  MeasuredValue(AttrType<Option<i16>>) = 0x0,
  MinMeasuredValue(AttrType<Option<i16>>) = 0x1,
  MaxMeasuredValue(AttrType<Option<i16>>) = 0x2,
}
attribute_enum!(Attributes);
pub const MEASURED_VAUE: Attribute = Attribute::new(
  AttributesDiscriminants::MeasuredValue as _,
  Access::RV,
  Quality::from_bits(Quality::NULLABLE.bits() | Quality::PERSISTENT.bits()).unwrap(),
);
pub const MIN_MEASURED_VAUE: Attribute = Attribute::new( /* 省略 */ );
pub const MAX_MEASURED_VAUE: Attribute = Attribute::new( /* 省略 */ );
pub const CLUSTER: Cluster<'static> = Cluster {
  id: ID as _,
  feature_map: 0,
  attributes: &[FEATURE_MAP, ATTRIBUTE_LIST, MEASURED_VAUE, MIN_MEASURED_VAUE, MAX_MEASURED_VAUE], // Attributeのリスト
  commands: &[],
};
pub struct TemperatureMeasurementCluster {
  data_ver: Dataver,
  temperature: Cell<Option<f32>>, // MeasuredValueで返す温度を保持する
}
impl TemperatureMeasurementCluster {
  // (省略)
  pub fn get(&self) -> Option<f32> {
    self.temperature.get()
  }

  pub fn set(&self, temperature: Option<f32>) { // センサから読み出したりした温度を設定する
    if self.temperature.get() != temperature {
      self.temperature.set(temperature);
      self.data_ver.changed();
    }
  }
  pub fn read(
    &self,
    _exchange: &Exchange,
    attr: &AttrDetails,
    encoder: AttrDataEncoder,
  ) -> Result<(), Error> {
    if let Some(writer) = encoder.with_dataver(self.data_ver.get())? {
      if attr.is_system() {
        CLUSTER.read(attr.attr_id, writer)
      } else {
        match attr.attr_id.try_into()? {
          Attributes::MeasuredValue(codec) => codec.encode(writer, self.temperature.get().map(|v| (v * 100.0).round() as i16)), // Measured Valueは 0.01℃単位なので100倍する
          Attributes::MinMeasuredValue(codec) => codec.encode(writer, None),
          Attributes::MaxMeasuredValue(codec) => codec.encode(writer, None),
        }
      }
    } else {
      Ok(())
    }
  }
}
```

基本的にアトリビュートの定義を追加し、クラスターのアトリビュートを読み書きする処理を追加するだけで簡単にクラスターを追加できます。全体のソースコードは GitHub のリポジトリ (https://github.com/ciniml/rs-matter-example) に置いてあります。

Alexa/Google Home/Apple Home での動作

当日のデモ用に Alexa/Google Home/Apple Home との接続を試みましたが、どれもうまく行きませんでした。Alexa はコミッショニングの途中でネットワークを設定するときに **ScanNetworks** コマンドを呼び出しますが、esp-idf-matter では現時点では実装されていないため失敗します。Google Home はコミッショニングの途中で Bluetooth から WiFi 経由の接続に切り替えるところでうまく先にすすまなくなり失敗します。Apple Home はコミッショニングは成功しますが、その後の通信がうまくいかず、応答なしとなってしまいます。実際の Matter コントローラと組み合わせるにはもうすこし esp-idf-matter や rs-matter の実装が成熟する必要があるようです。

筆者について

- Kenta Ida
- twitter: [@ciniml]
- GitHub: https://github.com/ciniml/
- Nature Remo というスマートホーム・デバイスのファームを書いてます。

esp-idf-matter と Rust でサクッと Matter デバイスを作ってみる

初版　2025 年 01 月 25 日

発行	Kenta IDA
著者	Kenta IDA (@ciniml)
連絡先	fuga@fugafuga.org
Twitter	@ciniml
GitHub	https://github.com/ciniml/